Title	ZDDによるパスの列挙 (計算機科学とアルゴリズムの数理的基礎とその応用)
Author(s)	川原, 純; 斎藤, 寿樹; 鈴木, 拡; 湊, 真一; 吉仲, 亮
Citation	数理解析研究所講究録 (2011), 1744: 35-41
Issue Date	2011-06
URL	http://hdl.handle.net/2433/170975
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
第1744巻 2011年 35-41

ZDDによるパスの列挙

川原 純 †,‡　斎藤 寿樹 †,‡　鈴木 拡 ‡　湊 真一 ‡,†　吉仲 亮 †,‡

† 科学技術振興機構 ERATO 湊離散構造処理系プロジェクト
‡ 北海道大学大学院情報科学研究科

概要

与えられたグラフ中のパスを列挙する問題は古典的な問題である．これまで，解を列挙するアルゴリズムや解の数を数えるアルゴリズムがいくつか提案されているが，一般に解の数は極めて多く，解の数を求める問題は #P-完全であるため，満足のいく高速な列挙アルゴリズムの実現は難しい．

近年 Knuth は，ZDD (Zero-suppressed Binary Decision Diagram) を用いた任意のグラフのある2点間を端点とするパスを列挙するアルゴリズムを提案した．ZDDとは集合族を圧縮して表現できるデータ構造である．グラフの経路を辺の集合と同一視することによって，経路の集合はZDDで表現することができる．提案されたアルゴリズムは，出力結果がパス集合のZDDによる圧縮表現であり，従来手法に比べ高速に動作する．本発表では，まずKnuthのアルゴリズムを紹介し，次にKnuthのアルゴリズムを基にした筆者らによる多項式回のZDDの基本演算でパスの列挙を行うアルゴリズムを提案する．これらの手法と既存手法との比較実験を行い，ZDDによるパスの列挙手法の利を論ずる．

1 序

与えられたグラフ中の特定2頂点間のパスやハミルトン閉路など，特定の性質を満たす経路の列挙問題は古典的な問題である．これまで，解の数を数えるアルゴリズム [1, 7] や解を列挙するアルゴリズム [6] がいくつか提案されているが，一般に解の数は極めて多く，多くの経路問題で解の数を求める問題は #P-完全であり，あるいは，解の存在問題は NP 完全であって，満足のいく高速な列挙アルゴリズムの実現は難しい．

グラフ中の経路を辺の集合と同一視することで，経路の集合を辺の集合族とみなせる．集合族を圧縮して表現するデータ構造として，**ZDD** (Zero-suppressed Binary Decision Diagram) が提案されている [4]. ZDDは膨大な集合族を効率よく圧縮できるのみならず，集合データに対する多様な演算もまた効率よく実行可能であるという特長があり，VLSI 設計やデータマイニング，ニューラルネットワーク上の計算等々に，幅広く応用されている．

最近 Knuth[3] は，ZDD 構造を用いて任意のグラフの任意の2点間を端点とするパスを列挙する高速アルゴリズムを提案した．本発表では，まずKnuthのアルゴリズムを紹介し，次にKnuthのアルゴリズムを基にしたアルゴリズムを提案する．このアルゴリズムは多項式回のZDDの基本演算でパスの列挙を行う．これらの手法と既存手法との比較実験を行い，ZDDによるパスの列挙手法の利を論ずる．

2 Zero-Suppressed Binary Decision Diagram

ZDD [4] は，有限集合の部分集合の集合を，出次数が2である非循環有向グラフによって表現する．語の混乱を避けるため，以降，全体集合の要素をアイテムと呼び，全体集合の部分集合を（アイテムの）組

合せと呼ぶ．あるZDDの表現する組合せ集合をデータと呼ぶ．ZDDではアイテム間に全順序を仮定し，それぞれの組合せは順序に従ってリストとして表現される．ZDDには**根**と呼ばれる特殊ノードが1つあり，ある組合せがデータに含まれるか否かは，根からノードを順に辿ることで決定する．各ノードはアイテムでラベルづけされており，そのアイテムが当該組合せに含まれるか否かに応じて，ノードからの二本の出力枝の一方を決定的に選択する．アイテムが含まれることを意味する出力枝を**HI枝**と呼び，他方を**LO枝**と呼ぶことにする．アイテム間の全順序に従って，上流ノードのアイテムラベルは常に下流ノードのアイテムラベルよりも真に小さい．ZDDは出力枝を持たないただ2つの底ノード $\top$ と $\bot$ を持ち，$\top$ に到達することは当該組合せがデータに含まれることを意味し，$\bot$ への到達は含まれないことを意味する．データの圧縮表現であるZDDは，次なる2つの簡約化規則を持つ：

- ラベル，HI枝，LO枝の全てが同じであるような異なる二ノードは存在しないものとする (ノードの一意性)，
- HI枝が $\bot$ を指すノードは陽に表現しない (Zero-suppress 規則)．

ノードの一意性，Zero-suppress 規則を満たさないデータ構造にあっても，組合せがデータに入っているかどうかは上述のノード間の遷移規則によって決定できる．このようなものを**非既約なZDD**と呼ぶ．単にZDDと言えば上記簡約化規則を満たす既約なものを意味する．Zero-suppress 規則によって陽に表現されなくなったノードの扱いについては注意が必要である．たとえば図1のZDDに表現されるデータには，組合せ abc は含まれない．なぜなら，a でラベルづけされた根ノードからHI枝に辿ったあとのノードのラベルは b ではなく c であり，ここには b でラベルづけされたノードが Zero-suppress 規則を適用されたと見ることができる．これは，組合せに b が含まれるならば $\bot$ ノードに到達すべきこと，そのような組合せはデータ中に存在しないことを意味する．換言すれば，データに含まれる組合せの全てのアイテムは，ひとつずつ順番にZDDのノードによって「確認」されなければならない．

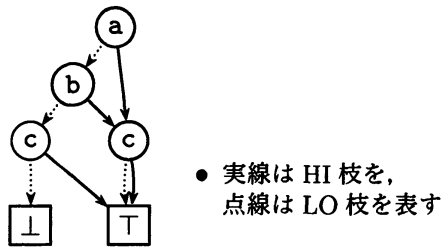

図 1:　集合 {a, b, c, ac, bc} を表す ZDD

データを大幅に圧縮表現するという特長に加え，もしくはそれ以上に重要なZDDの特長に，演算処理の容易性・効率性がある [5]．集合に新しい組み合わせを加える，組合せのうち特定のアイテムを含む／含まないもののみを抜き出す，各アイテムに重みを仮定して最大重みを与える組合せを求めること，等々が容易に可能である．ここで，ZDDの表す集合 F を多項式で表すと便宜だろう．例えば図1では $F = \mathtt{a} + \mathtt{ac} + \mathtt{b} + \mathtt{bc} + \mathtt{c}$ となる．この例を用いて，順に，和，積，商，剰余をとる演算を紹介する．

- 足算 $+$：集合 F に組合せ abc を追加するときは $F \leftarrow F + \mathtt{abc}$ と命令する．結果は $F = \mathtt{a} + \mathtt{abc} + \mathtt{ac} + \mathtt{b} + \mathtt{bc} + \mathtt{c}$ となる．
- 掛算 $\times$：集合 F 中の全ての組合せにアイテム d を追加するときは $F \leftarrow F \times \mathtt{d}$ と命令する．結果は $F = \mathtt{ad} + \mathtt{abcd} + \mathtt{acd} + \mathtt{bcd} + \mathtt{bd} + \mathtt{cd}$ となる．
- 割算 $/$：F の全ての組合せのうち，特定のアイテム（の組合せ）を含むものについて，それを取り除いた結果を与える．$F \leftarrow F/\mathtt{a}$ の結果は $F = \mathtt{bcd} + \mathtt{cd} + \mathtt{d}$ である．
- 剰余算 $\%$：F の全ての組合せのうち特定のアイテム（の組合せ）を含まないもののみを与える．$F \leftarrow F \,\%\, \mathtt{c}$ の結果は $F = \mathtt{d}$ となる．

これらの演算は2つの異なる集合 F, G を表す ZDD 間でも同様に実行可能である．ZDD ではこれらの演算をある種の因数分解と対応したままで実行するため，式を展開して行うよりも高速に演算できる．上記の基本演算以外にもさらに多様な演算が実装・提供されている．

3 ZDD によるパスの列挙アルゴリズム

グラフ $G = (V, E)$ において，$i \in \{1, 2, \ldots, l-1\}$ に対し $\{v_i, v_{i+1}\} \in E$ であるとき，頂点の列 $(v_1, v_2, \ldots, v_l)$，ただし $i \neq j$ で $v_i \neq v_j$，を v_1 から v_l への**パス**という．また，パス $(v_1, v_2, \ldots, v_l)$ に対し，辺の集合 $\{\{v_i, v_{i+1}\} \mid i \in \{1, \ldots, l-1\}\}$ をパスと同一視する．互いに素なパスの集合を**パスマッチング**という．

本論文では，以下の問題を考える．

入力：グラフ $G = (V, E)$，2頂点 s と t

出力：s から t へのすべてのパスを格納した ZDD．

本論文では次のような ZDD でグラフ G 上のパスを格納する．グラフ G の各辺 $e \in E$ は ZDD におけるアイテムであり，各 s から t へのパス P は ZDD 上の根から $\top$ までのあるパスに対応し，P に含まれる辺は HI 枝を通り，P に含まれない辺は LO 枝を通る．

本節では，この ZDD を構築する2つのアルゴリズムについて述べる．1つ目は近年，Knuth により提案されたアルゴリズムで [3]，このアルゴリズムを実装した SIMPATH と呼ばれるプログラムが公開されている．2つ目は本論文で提案するアルゴリズムで，SIMPATH のアルゴリズムを元にした，ZDD で行える多項式回の基本演算のみを用いたアルゴリズムである．

3.1 Simpath のアルゴリズム

SIMPATH のアルゴリズムの主なアイディアはパスマッチングの集合 $\mathcal{P}$ を管理することである．具体的には，まず，初期化として $\mathcal{P}$ を空のパスマッチング1つからなる集合とする．また，辺に順序が与えられており，各辺で次の操作が行われる．$\mathcal{P}$ の各パスマッチング P について，P に辺 e を追加したパスマッチング P' および e を追加しないパスマッチングを $\mathcal{P}_{\mathrm{new}}$ に格納する．$\mathcal{P}$ の全要素の処理の終了後，$\mathcal{P}_{\mathrm{new}}$ の全要素を $\mathcal{P}$ に移動させる．すべての辺への処理が終了したとき，s から t へのパスのみで構成されるパスマッチングが出力対象である．しかし，単純にこれを実行すると，パスマッチングの数は非常に多いため，膨大な計算時間および領域が必要となる．

SIMPATH では，すべてのパスマッチングを管理せず，今後行う操作が同じであるパスマッチングを同一視し，管理すべきパスマッチングの数を減らしている．ここで，$E' \subset E$ を現在までに操作した辺の集合とし，$P, P' \subset E'$ をパスマッチングとする．このとき，任意の $P'' \subset E \setminus E'$ に対し，$P \cup P''$ が s から t へのパスであるとき，またそのときに限り，$P' \cup P''$ が s から t へのパスであるならば，P と P' を**同値なパスマッチング**という．SIMPATH では，mate と呼ばれる配列を用いて，同値であることが明らかなパスマッチングを1つにまとめて取り扱う．配列 mate は要素数 $|V|$ で配列の添字は頂点に対応する．mate の各要素の値はパスマッチング P の状態によって，以下のような値を取る．

$$\mathrm{mate}_P[i] = \begin{cases} j & P \text{に} i, j \text{を端点とするパスが存在} \\ i & \text{頂点} i \text{の接続辺が使われていない} \\ 0 & \text{頂点} i \text{は} P \text{のあるパスの中点} \end{cases}$$

mate が同一なら，2つのパスマッチングは同値である．

SIMPATH のアルゴリズムは非既約な ZDD を先に構築し，後から非既約な ZDD の簡約化を行い，解となる ZDD を出力する（非既約な ZDD の簡約化については文献 [3] を参照）．SIMPATH の作成する ZDD のノードは，mate，LO 枝，HI 枝をもつ．ノード N の LO 枝（HI 枝）が指すノードを $LO(N)$（$HI(N)$）と表記する．SIMPATH の非既約な ZDD 構築部分のアウトラインは以下の通りである．

1. $\mathcal{P} := \{N_0\}$, $\mathcal{P}_{\text{new}} := \emptyset$
   /* N_0 は初期ノード（空のパスマッチング） */
2. **For each** 辺 $e \in E$ **do**
3. **For each** ノード $N \in \mathcal{P}$ **do**
4. N に e を加えたノードを N_{HI} とする.
5. N に e を加えないノードを N_{LO} とする.
6. **For** $X = HI, LO$ **do**
7. **If** N_X が $\mathcal{P}_{\text{new}}$ のいずれかの要素 N' と「同値」 **then** $N_X := N'$
8. **else** N_X を $\mathcal{P}_{\text{new}}$ に加える.
9. **If** N_X の mate が 1 つの s-t パスを表す **then** $X(N) := \top$
10. **else if** N_X の mate が無効 **then** $X(N) := \bot$
11. **else** $X(N) := N_X$.
12. **End For**
13. **End For**
14. $\mathcal{P} := \mathcal{P}_{\text{new}}$, $\mathcal{P}_{\text{new}}$ を空にする.
15. **End For**

4. では N の表すパスマッチングに辺 $e = \{u, v\}$ を追加して新しいパスマッチングを生成する. 実際には, N はパスマッチングそのものではなく, mate を保持していることに注意せよ. 具体的には辺 e を追加したとき, mate を次のように更新する. $\text{mate}[u] = w$ かつ $\text{mate}[v] = x$ のとき, $\text{mate}[x] = w$, $\text{mate}[w] = x$ とする. このとき, $v \neq x$ ならば $\text{mate}[v] = 0$, $u \neq w$ ならば $\text{mate}[u] = 0$ とする.

9. では mate が 1 つの s-t パスを表すかどうか判定する. 具体的には $\text{mate}[s] = t$ かつ $\text{mate}[t] = s$ かつ, それ以外の $\text{mate}[i]$ が $\text{mate}[i] = 0$ or i であるならば, mate が表すパスマッチングは 1 つの s-t パスである. このとき, HI 枝の指す先を $\top$ に設定する. 9. 10. 11. の $X(N) := N'$ は N の HI 枝（LO 枝）の指すノードを N' に設定するという意味である.

10. では, mate が「無効」であるか判定を行う. mate が無効であるとは, 以下の (i), (ii) のいずれかが成り立つことと定義する. (i) 頂点の次数が 3 以上になる等, mate がパスマッチングになっていない. (ii) 頂点 v ($v \neq s$ かつ $v \neq t$) に接続するすべての辺

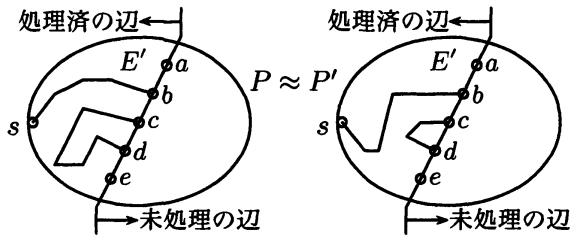

図 2: 頂点 a, e が孤立し, s–b, c–d がパスをなす 2 つの同値なパスマッチング P と P'

の処理が終ったとき, v の次数が 1 である. (i) は, $\text{mate}[u] = v$ ($\text{mate}[v] = u$), または $\text{mate}[u] = 0$, または $\text{mate}[v] = 0$ のとき mate は無効であると判定する. (ii) は, $\text{mate}[v] = v$ または $\text{mate}[v] = 0$ となっていない場合, mate は無効であると判定する.

7. ではノードの同値性判定を行う. 多くの同値なパスマッチングを同一視するために, フロンティアだけ mate を抜き出し, 同値判定に用いる. フロンティアとは少なくとも 1 つの処理済の辺 ($\in E'$) および少なくとも 1 つの未処理の辺 ($\in E \setminus E'$) の両方と接続している頂点の集合である (図 2). 例えば, 図 2 の 2 つのパスマッチングは mate 全体の値は異なるが, フロンティア上の mate の値が同一であるので, 同値である. このように, 処理済のどの辺をどのように使っているかは関係なく, フロンティア上の頂点の mate の値が等しい 2 つのパスマッチングは同値であるため, フロンティア上の頂点の mate のみを管理すればよい. これにより, 非既約な ZDD のノード数を削減し, 計算時間および領域を大幅に短縮することができる.

3.2 Bottom-up Mate-ZDD 法

本節では, Bottom-up MATE-ZDD 法（もしくは, MATE-ZDD 法）というアルゴリズムを提案する. このアルゴリズムのベースとなるアイディアは SIMPATH のアルゴリズムと同様である. 決定的な違いは, SIMPATH は mate を管理し (非既約な)ZDD を構築するが, MATE-ZDD 法は ZDD で行える基本演算（足算, 掛算, 割算, 剰余算）のみを用いて, ZDD

を更新していくアルゴリズムである.

MATE-ZDD 法は，パスマッチングを保持する ZDD F と，さらにフロンティア上の mate 配列の代わりとなる補助変数を管理する．具体的には，$\mathrm{mate}[i] = j$ $(j \neq 0)$ に対し，MATE-ZDD 法では補助変数 $t_{i,j}$ が対応し，変数 $t_{i,j}$ のノードの HI 枝は i と j を両端点とするパスを持つパスマッチングに対応する.

MATE-ZDD 法のアルゴリズムの流れは以下のとおりである．まず，F の初期値を $F = t_{1,1} \ldots t_{n,n}$ (頂点を $1, \ldots, n$ とする) とする．決められた順番に辺を処理していき，mate とほぼ同様の更新を ZDD の演算を用いて行う．アルゴリズムが処理済みの辺を $E' \subset E$ とし，パスマッチング集合 $\mathcal{P}$ を $\mathcal{P} := \{P \mid P \subset E'\}$ とする．V' をフロンティアとする．アルゴリズムの途中で F は以下の形となる.

$$F = \sum_{P \in \mathcal{P}} F_P,$$

ただし，

$$F_P = \prod_{\substack{i,j \in V',\ i \leq j \\ \mathrm{mate}_P[i] = j}} t_{i,j} \prod_{\{k,\ell\} \in P} e_{k,\ell}.$$

各 F_P は 1 つのパスマッチング P に対応する．e 変数の積で 1 つのパスマッチング P を表現し，t 変数の積でパスマッチング P の mate を表現する.

辺 $\{u, v\}$ を追加するときの更新式は以下の通りである．各 $w, x \in V'$ に対し，

$$F = F + (F/t_{u,w}/t_{v,x}) \times e_{u,v} \times t_{x,w}$$

とする．また，頂点 v がフロンティアから抜けるとき，$v \neq s, t$ ならば v の次数は 1 であってはならない．そのため，フロンティアの各頂点 $w \neq v$ に対し，

$$F = F \,\%\, t_{v,w}$$

を行う．また，頂点 v が孤立しているかどうかに関心はないので

$$F = F/t_{v,v} + F \,\%\, t_{v,v}$$

とする．そして，最終的に構築した ZDD F に対し，$t_{s,t}$ を含んだ項 $F = F/t_{s,t}$ が求める ZDD である.

グラフ	CyPath	Simp.	M.-ZDD	ノード数	パスの数
JM	> 1000	< 0.01	< 0.01	850	5.1×10^{10}
JM^2	> 1000	24.01	248.62	1.7×10^7	5.5×10^{43}

表 1: JapanMap (JM) と JapanMap^2 (JM^2) 上のパスの列挙（単位：秒）.

4 実験結果

MATE-ZDD アルゴリズムの実装を行い，Knuth [3] による SIMPATH プログラムと比較を行った．また，Read と Tarjan による Backtrack のアルゴリズム [6] (宇野毅明氏による実装である CYPATH [8] プログラム) とも比較を行った．JapanMap は都道府県を頂点とし，都道府県が地図上で隣接している (歩行，車，鉄道のいずれかで移動可能な) 場合に頂点間に辺を張ることによって生成したグラフである．沖縄は除くので頂点数は 46 である．JapanMap^2 は JapanMap の頂点と辺をそれぞれ二重化したグラフである．これらのグラフに対し，各プログラムを用いて，始点を北海道，終点を鹿児島としたパスの列挙を行ったのが表 1 である．表中の Simp. は SIMPATH による計算時間を，M.-ZDD は MATE-ZDD による計算時間を意味する．計算環境は CPU が AMD Quad-Core Opteron 8393 SE (3.09GHz)，メモリが 512GB である.

MATE-ZDD プログラムが出力する ZDD のノード数は，変数の順序に大きく依存する．今回の実験のための実装では，ノード数が小さくなるようにヒューリスティックに変数順序を決定している．表 1 の「ノード数」の項目は，この変数順序の元で MATE-ZDD プログラムが出力した (既約な) ZDD のノード数を意味する．JapanMap^2 上のパス列挙において，生成された ZDD のノード数は 17,036,796 個であり，一方，パスの数は 552,990,153,767,713,569,683,921,601,890,579,271,385,088 個である．ZDD によって膨大な数のパス集合をコンパクトに表現できていることが分かる.

ランダムグラフについても，各プログラムを用いてパスの列挙を行った（表 2）．頂点数 n を指定し，

n	CyPath	Simp.	M.-ZDD	ノード数	パスの数
15	3.33	0.29	2.87	1.5×10^5	3.7×10^6
16	25.20	0.96	14.84	6.2×10^5	2.9×10^7
17	147.62	2.99	64.45	2.2×10^6	1.6×10^8

表 2: ランダムグラフ上のパスの列挙，辺の生成確率 0.5（単位：秒）

n	CyPath	Simp.	M.-ZDD	ノード数	パスの数
8	> 1000	0.03	0.44	31487	7.8×10^{11}
9	> 1000	0.15	1.92	110215	3.2×10^{15}
10	> 1000	0.63	6.00	377202	4.1×10^{19}
11	> 1000	1.88	23.75	1.2×10^6	1.5×10^{24}
12	> 1000	6.25	148.11	4.2×10^6	1.8×10^{29}

表 3: 格子グラフ上のパスの列挙（単位：秒）

辺の生成確率を $p = 0.5$ とし，グラフを 100 個生成して列挙時間の平均を求めた．$n = 15$, 16, 17 の場合に，3 つのプログラムの計算時間の差が明確に表れた．

$n \times n$ の格子グラフ（グリッドグラフ）及び n 頂点完全グラフについても，各プログラムを用いてパスの列挙を行った．結果を表 3, 4 に示す．

5 議論

本研究では Knuth の ZDD によるパス列挙プログラム SIMPATH と，それを基に独自に提案した Bottom-up MATE-ZDD 法を比較検討した．圧縮した表現でパスを列挙するため，両手法とも，既存の列挙手法 (CYPATH [8]) に比して著しく高速に動作する．さらに，これらのプログラムはわずかな変更で，ハミルトン閉路の列挙等への応用が可能である．

これら ZDD に基づくプログラムの比較においては MATE-ZDD 法が SIMPATH に比して一桁遅いが，この理由として (1) SIMPATH は非既約な ZDD を構築し，最後にまとめて簡約化を行うのに対し，MATE-ZDD は計算途中の構築物が常に既約な ZDD であるように維持していること，(2) SIMPATH で 1 つの mate に相当するものが，複数の t 変数を木状に配置

n	CyPath	Simp.	M.-ZDD	ノード数	パスの数
11	0.58	0.03	0.32	33830	9.9×10^5
12	5.79	0.07	1.34	121455	9.9×10^6
13	64.30	0.42	8.51	435810	1.1×10^8
14	770.09	0.92	25.40	1.6×10^6	1.3×10^9
15	10049.51	2.74	110.07	5.6×10^6	1.7×10^{10}

表 4: 完全グラフ上のパスの列挙（単位：秒）

した物によって管理されるため，探索において，その木の登り降り等，その更新に時間がかかること，などが考えられる．一方で，MATE-ZDD 法はアルゴリズムが単純で，メンテナンス性に優れる．例えば，2 つの離れた辺の間に共起性や排他性のような関係がある場合にこれを実現することが容易である．常に正しく ZDD を構築しているので，ある辺を追加するか否かを決めるときに，共起すべき辺が構築中のパスに含まれるかどうかに応じた挙動を取ることが容易に実装できる．これは SIMPATH にあっては mate の概念を拡張することなしには実現できず，条件が複雑化したときの実装が難しくなる可能性がある．もっとも，条件を無視して ZDD でパスを列挙した上で，条件を満たすものを ZDD 演算で取り出すこともできるので，一概に計算速度について論じることは難しい．今後の研究課題としたい．

また，グラフにおける頂点の探索順序あるいは辺に対応する ZDD の変数順序が，計算時間や構築される ZDD のサイズに影響することがわかっている．現在は開始地点から幅優先で決めているが，これが最適ではない例が存在することがわかっている．グラフの特徴に応じたより良い頂点の探索順序の決定手法も今後の課題である．

参考文献

[1] E. T. Bax. Algorithms to Count Paths and Cycles. *Information Processing Letters*, 52, pp. 249-252, 1994.

[2] M. B. Javanbarg, C. Scawthorn, J. Kiyono, and Y. Ono. Minimal Path Sets Seismic Reliability

Evaluation of Lifeline Networks with Link and Node Failures. In *Proc. Lifeline Earthquake Engineering in a Multihazard Environment.*

[3] D. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1.*

[4] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *DAC*, pp. 272-277, 1993.

[5] S. Minato. Fast Factorization Method for Implicit Cube Set Representation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, VOL. 15, No. 4, pp. 377-384, 1996.

[6] R. C. Read and R. E. Tarjan. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5, pp. 237-252, 1975.

[7] K. Sekine and H. Imai. Counting the Number of Paths in a Graph via BDDs. *IEICE TRANS. FUNDAMENTALS*, VOL. E80-A(4), pp. 682-688, 1997.

[8] T. Uno. CyPath: *st* path, cycle, chordless *st* path, chordless cycle enumeration. http://research.nii.ac.jp/~uno/code/cypath.html.